Exame de Linguagens de Programação 2

22 de Julho de 2002 (Duração: 2H30M)

I

1. [2] Indique o valor dos programas Oby seguintes:

   (a)
   ```
   let dup = fun f -> fun x -> f(f(x))
       in dup(fun x -> x+2)(3)
   ```

   (b)
   ```
   letrec f = fun x -> f(x)
       in let K = fun x,y -> x
           in K(2, f(2))
   ```

2. [2] Considere os programas acima:

   (a) Explique detalhadamente o funcionamento do programa 1(a), apresentando o valor do ambiente no momento em que se inicia a avaliação da expressão `x+2`.

   (b) Indique qual o valor do programa 1(b) se em vez da regra de avaliação de parâmetros *call-by-value* usual for adoptada a regra *call-by-name*.

II

Considere a linguagem Oby estudada nas aulas. A linguagem Oby é muito poderosa apesar de muito simples, pois permite "programar" praticamente todas as construções das linguagens de programação modernas. Se em vez de usar passagem de argumentos *call-by-value*, usasse a regra *call-by-name*, seria até possível definir com facilidade todas as construções de controle das linguagens de programação imperativas.

1. [2] Vamos usar nesta alínea a linguagem ObyCBN, que é uma linguagem em tudo idêntica à linguagem Oby, mas que adopta a regra *call-by-name* em vez da regra *call-by-value*. Considere a construção

   `for(`E_1 `;`E_2`;` E_3`)` S

   da linguagem `ANSI C`. Escreva uma função ObyCBN `for_impl`, com quatro parâmetros (E_1, E_2, E_3 e S), que permita implementar a construção `for`.

2. [1] Explique porque não é possível escrever directamente a função `for_impl` na linguagem Oby que usa a regra de passagem de argumentos `call-by-value`.

3. [2] Em linguagens com passagem de argumentos *call-by-value* é possível "simular" a passagem de argumentos *call-by-name* do seguinte modo: em vez de passar como argumento para o parâmetro `X` a expressão `E`, passamos como argumento uma abstracção `(fun -> E)`. Dentro do procedimento, em vez de se usar o parâmetro `X` directamente, escreve-se `X()` para obter o seu valor. Usando esta técnica, traduza o seguinte programa ObyCBN num programa Oby que simule a passagem de argumentos *call-by-name*.

   ```
   let x = new 1
     in let exec_twice = fun X -> { X; X}
          in { exec_twice(x:=!x+1); !x }
   ```

   Nota: usando a regra *call-by-name*, este programa deverá produzir como resultado o valor 3; usando a regra *call-by-value* (a regra usada na linguagem Oby), o valor produzido será 2.

III

A maior parte das linguagens de programação permitem definir valores obtidos não apenas por agregação de um tuplo de valores mas também por selecção de um valor de entre um conjunto de alternativas. Por exemplo, temos os tipos enumerados em C++, os tipos soma da linguagem ML, os registos com variante da linguagem Pascal, etc. Este exercício aborda programas *object-oriented* escritos na linguagem ObyS, que é um extensão da linguagem Oby que inclui tipos soma **em vez de registos**. Os valores de tipo soma chamam-se "alternativas". Uma *alternativa* é uma expressão da forma

$$id\{E_1, E_2, \ldots, E_n\}$$

onde id é uma etiqueta (um identificador) e os E_i são expressões quaisquer da linguagem ObyS. Eis alguns exemplos de alternativas:

```
nil{}
cons{1,nil{}}
node{x,1+2,y}
```

A única operação definida sobre alternativas é a construção de análise de casos `case`, que é uma expressão da forma:

$$\begin{array}{l}\texttt{case } E \texttt{ of}\\ \quad l_1\{x_1^1, \ldots, x_{n_1}^1\} = E_1\\ \quad \cdots\\ \quad l_k\{x_1^k, \ldots, x_{n_k}^k\} = E_k\\ \texttt{end}\end{array}$$

Os identificadores $x_1^i, \ldots, x_{n_i}^i$ são ocorrências ligantes de variáveis (cujo âmbito abrange cada expressão E_i), os l_i são etiquetas, e E e os E_i são expressões quaisquer.

A semântica da construção `case` é a seguinte: primeiro, é avaliada a expressão E, que deve produzir como resultado uma alternativa construída com base numa das etiquetas $l_1, \cdots, l_k$ (se assim não for, deverá ocorrer um erro de execução). Assim, podemos supor que o valor de E é a alternativa $l_j\{v_1, \cdots, v_m\}$, sendo l_j a etiqueta da alternativa associada à expressão E_j. Seguidamente, será avaliada a expressão E_j num contexto (ambiente) onde o valor de cada identificador x_i^j estará associado ao valor v_i existente na posição correspondente da alternativa. O resultado de tal avaliação será o valor devolvido finalmente pela expressão `case`.

Por exemplo, usando as alternativas `nil{}` e `cons{a,b}` como construtores podemos definir listas: `cons{1,cons{2,nil{}}}` representa a lista dos dois inteiros 1 e 2. E uma função que soma os elementos de listas representadas dessa forma por

```
letrec sum = fun l ->
  case l of
    nil{} = 0
    cons{x,y} = x+sum(y)
  end
```

1. [2] Usando valores alternativa podemos representar os vários métodos a que um objecto pode responder. Mostre como pode representar um objecto "contador" em ObyS. (Sugestão: represente um objecto por uma função que recebe uma mensagem (método) como argumento, e utiliza a construção `case` para seleccionar o código a executar).

2. [2] Usando a técnica da alínea anterior, defina em ObyS uma classe `counter` de objectos contadores, usando a técnica da função geradora.

3. [2] Uma classe derivada pode ser implementada em ObyS através de uma função que quando aplicada a um objecto da classe base, devolve um novo objecto. Ilustre este mecanismo escrevendo uma classe `dupcounter` que deriva da classe `counter` definida em 2. A classe `dupcounter` deverá implementar apenas um método, definido como

   ```
   dup{ } = { this(inc{}); this(inc{}) }
   ```

   e herdar os métodos definidos na classe `counter`.